Morphology based OCR

Hugo Woolf

Abstract

This paper discusses the development of an optical character recognition (OCR) technique that is intended to recognize a person's handwriting. The technique has evolved over time, the various evolutions and improvements are discussed in this paper.

Introduction

OCR has many real-world uses. There are many tasks that can be automated to save money that would normally be used to pay employees to spend hours doing menial tasks. In some cases, companies keep huge amounts of paper-based records. In most cases, the company cannot afford to hire people to digitize the records, or they just can't be bothered to do it because it takes a lot of time and work. By using OCR software, the process can be completed in a fraction of the time with a fraction of the work and cost.

Besides business applications, OCR can be useful to the average person. Sometimes it's just inconvenient to bring a laptop (or other electronic device) along to take notes or keep records. Typing up these things is often not worth it to the user. With OCR software, it can be as easy as just scanning the paper and running a program to get a digital version of your notes.

Background+Development

The image processing techniques used in this project are all relatively simple. The first set of techniques explored were edge detection methods. Specifically, the Sobel and Canny operator. The input is blurred using a Gaussian blur before either edge detection method is applied. This ensures that the edges will be uniformly free of spurs and connected as best as possible.

The Sobel method is very simple, it is simply a function of the gradient of the image at every pixel, the higher the gradient, the more likely a pixel is marked as an edge. This produces results that are very good, when viewed by a human, and when obtained using an appropriate threshold. Unfortunately, this threshold value is hard to find, and is unique for every image, so this method is not very useful in practice. The Canny method makes an improvement upon this, by using two thresholds and recursively marking off pixels that meet the low threshold by may not have met the high threshold. These method tend to be a bit slow for large images, and are not entirely practical on scanned image, which are usually high resolution. For the final product, I decided to use the most basic method, a simple threshold. It's effective enough for the intended purposes, provided that the input is high contrast.

The original method used canny edge detection to reduce the image to single-pixel-wide lines. There were various methods applied to the canny output to detect corners and other features. One method was the addition of the distances of all the points in a square around the working point. In other words, if a point lies on a corner, this value will be much higher than if it were to lie on a line. Points on a line will yield a value of zero. This method works fairly well, and may be used in the final project.

Currently, morphological operators are use exclusively for all steps of the process. After thresholding, the image is thinned by passing these matrices, and their 90 degree rotations over the image.

0	0	0
	1	
1	1	1

	0	0
1	1	0
	1	

If a 3x3 block matches the matrix, the center pixel is set to 0. By iteratively applying this to a binary matrix until convergence, one can thin an image.

Similar matrices are used for further processing. Simply thinning the image results in a skeleton. While this is only a single pixel wide, the branches and spurs can be very detrimental to image processing. In the figure

below, the top sample is before pruning, the bottom is after.

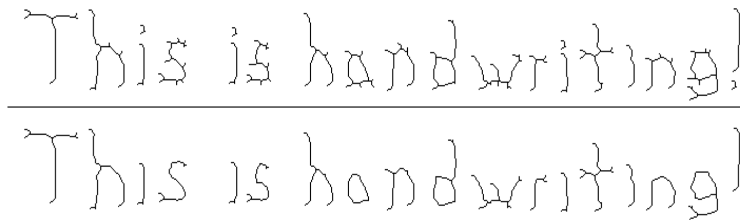

Fig 1.2, output before and after pruning.

The current pruning method consists of several steps, the first of which is a simple prune using these matrices:

0	0	0
0	1	0
0		

0	0	0
0	1	0
		0

After pruning a certain number of iterations, the pruned image is dilated and intersected with the original image, the result can be seen in fig 1.2 above.

By doing simple pruning until convergence, only the closed loops in an image remain. This will likely be useful information, as there are a number of letters that contain closed loops, being able to differentiate between those letters and the rest of the alphabet will be useful.

Conclusion

The final step in the process is identifying the output as individual characters. For this, a neural network implementation was used. The training data was obtained by thinning and pruning various handwriting images and identifying the letters by hand. The pool of training data was likely not nearly large enough to successfully train the network. The network consisted of 500 nodes per layer and 2 layers, it produces 26 output which are all either 0 or 1. Due to various issues, the network was never fully trained and its success rate could not be accurately determined. Ideally, after identification, the characters would be reassembled into their words. The resulting words would then be compared with a dictionary to further increase the accuracy of the recognition. Characters that were marked as multiple letters or as no letter could be fixed by finding the word in a dictionary and filling in the blanks.

Overall, this technique seems to be fairly effective if implemented properly. It seems the most difficult step is properly training the neural network.